Finite Automata

Last Updated: February 1 st, 2023

1 Introduction

The finite automaton perhaps represents the most basic yet nontrivial model of computation. A finite automaton consists of a finite number of states which serve as the automaton's only source of memory. Since an automaton is capable of accepting languages having arbitrarily long words, it must read each input word (i.e. problem instance) symbol by symbol until all symbols have been read at which point it either accepts or rejects w.

Finite automata have applications in several areas of computing, including computer architecture, cellular automata, natural language processing and recognition, compiler theory, and text processing to name just a few. Finite automata solve problems that require only a finite amount of memory, regardless of the problem size.

A **deterministic finite automaton (DFA)** consists of a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

Q is a finite set with each member of Q called a **state**

Σ a finite alphabet that is used to represent an input word

δ a transition function that determines the next state of the automaton given the current state and the current input symbol being read. In other words,

$$\delta : Q \times \Sigma \to Q,$$

is a mapping from (current) state-input pairs to (next) states.

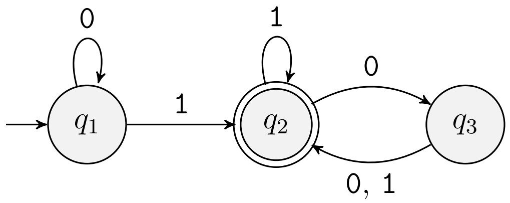
$q_0 \in Q$ the **initial state**.

$F \subseteq Q$ a member of F is called an **accepting state**.

Note that a DFA is capable of reading each input symbol only once, and in the order in which it appears in the input word w.

A graphical way of representing a DFA is to represent the states with graph nodes, with an arrow pointing at the initial state, and accepting states circled. Furthermore, a directed edge labeled with s starts at node q_i and terminates at node q_j iff $\delta(q_i, s) = q_j$.

Example 1. Provide the 5-tuple for the following DFA.



1.1 DFA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and $w = w_1 w_2 \cdots w_n$ be a word in Σ^*. Then the **computation of M on input** w is a sequence $q_0, q_1, q_2, \ldots, q_n$ of states recursively defined by

1. q_0 is the initial state of M; and
2. $q_i = \delta(q_{i-1}, w_i)$, for every $1 \leq i \leq n$.

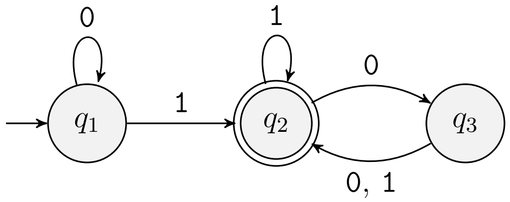
In words, the computation of M on input w is the sequence of states $q_0, q_1, q_2, \ldots, q_n$ that M moves through when successively reading input symbols $w_1, w_2, \ldots, w_n$. The computation of M on input w is said to be **accepting** iff $q_n \in F$. In this case we say M **accepts** w. Otherwise it is called a **rejecting** computation, and M **rejects** w.

One way to follow a computation is to start off by placing a pebble in initial state q_0. Then every time an input symbol s is read, pick up the pebble from its current state location q and place it in state $\delta(q, s)$. Continue doing this until each symbol of the entire input word has been read.

Note: a DFA-computation simulator can be found at

`https://www.cs.cmu.edu/~cburch/survey/dfa/index.html`

Example 2. For the DFA M shown below, provide the computation of M on input i) $w_1 = 01101011$ and input ii) $w_2 = 0011000$.



1.2 Regular language

Given DFA M, the **language accepted by** M is defined as

$$L(M) = \{w \in \Sigma^* | M \text{ accepts } w\}.$$

If a language $L \subset \Sigma^*$ is accepted by some DFA M, then L is called a **regular language**.

Example 3. Let L be the set of binary strings w such that every odd bit of w equals 1. Show that L is a regular language.

Example 4. Provide a succinct description of the language that is accepted by the DFA shown in Example 1.

Example 5. Provide the state diagram for a DFA that accepts the language of all binary strings that begin with two 0's and have an odd number of 1's.

Example 6. Provide the state diagram for a DFA that accepts the language of all binary strings that contain the substring 101.

2 Nondeterministic Finite Automata

To prove the closure of regular languages under union, concatenation, and star, it will be of great help to use the concept of a nondeterministic finite automata (NFA). An NFA has almost the same definition as a DFA, with one exception: upon reading an input symbol, a state transition allows for more than one possible next state. More formally, the codomain of function δ is now longer Q, but rather $\mathcal{P}(Q)$, the set of all subsets of Q.

A **nondeterministic finite automaton (NFA)** consists of a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

Q is a finite set with each member of Q called a **state**

Σ a finite alphabet that is used to represent an input word

δ a transition function that determines the set of possible next states of the automaton given the current state and the current input symbol being read. In other words,

$$\delta : Q \times \Sigma \to \mathcal{P}(Q),$$

is a mapping from (current) state-input pairs to a *subset* of possible next states.

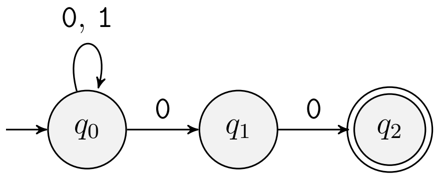
$q_0 \in Q$ the **initial state**.

$F \subseteq Q$ a member of F is called an **accepting state**.

Note that every DFA is technically an NFA for which the transition function maps each state-input pair to a subset of size 1. Indeed, DFA transition function statement $\delta(q, s) = r$ can be expressed like an NFA statement by writing

$$\delta(q, s) = \{r\} \in \mathcal{P}(Q).$$

Example 7. The following state diagram provides an example of an NFA N that is not a DFA, since state q_0 has two different outgoing edges that are labeled with 0. Notice that states q_1 and q_2 are both missing outgoing edges. This is done to simplify the state diagram. The convention is, if N is in state q, reading input symbol s, and there is no outgoing edge from state q that is labeled with s, then the default is $\delta(q, s) = \emptyset$.



The following table represents transition function δ.

$q_i \backslash s$	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\emptyset$
q_2	$\emptyset$	$\emptyset$

2.1 NFA computation

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA, and $w = w_1 w_2 \cdots w_n$ be a word in Σ^*. Then the **computation of N on input** w is a sequence $S_0, S_1, S_2, \ldots, S_n$ of state subsets recursively defined by

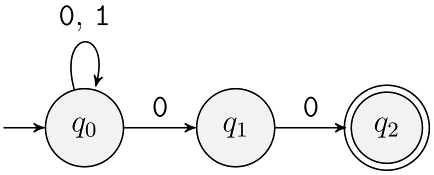
1. $S_0 = \{q_0\}$ is the singleton set consisting of the initial state q_0 of M; and
2. $S_i = \bigcup_{q \in S_{i-1}} \delta(q, w_i)$, for every $1 \leq i \leq n$.

In words, the computation of N on input w is the sequence of state sets $S_0, S_1, S_2, \ldots, S_n$ that N moves through when successively reading input symbols $w_1, w_2, \ldots, w_n$. The computation of N on input w is said to be **accepting** iff $S_n \cap F \neq \emptyset$. In other words, at least one accepting state belongs in S_n. In this case we say N **accepts** w. Otherwise it is called a **rejecting** computation, and N **rejects** w.

Rather than say that an NFA is in a particular state, we instead say that it is in a particular subset S states. So, during an NFA computation, rather than having a single pebble in some state S, there are instead multiple pebbles, one pebble per state $q \in S$. Then when an input symbol s is read, those pebbles are replaced with new ones, one for each state in

$$\bigcup_{q \in S} \delta(q, s).$$

Example 8. For the NFA N shown below, show the computation of N on inputs 010100 and 11010.

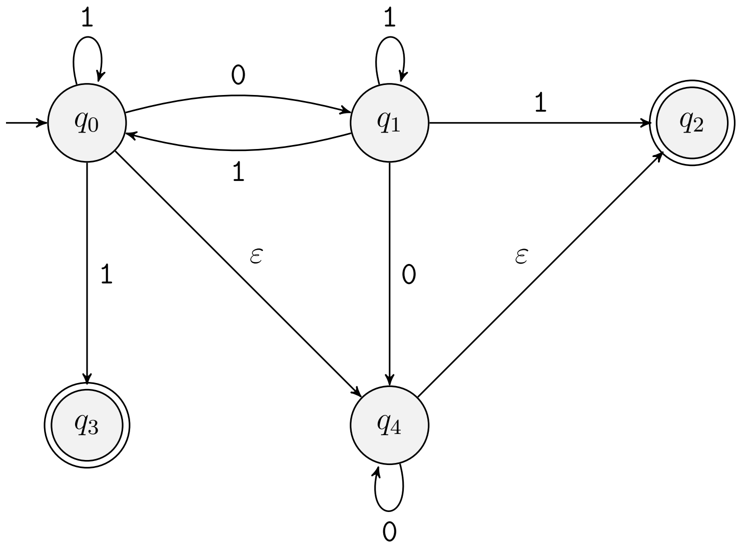


2.2 ε transitions

Another difference between an NFA state diagram and a DFA diagram is that an NFA node may have an edge leaving it that is labeled with ε. This means the following statements are true.

1. If there is an ε-labeled edge from q_1 and q_2, then q_2 is in a state subset S of the transition table whenever $q_1 \in S$.
2. During a computation, if a pebble is placed on q_1 at any time, then one must also be placed on q_2.
3. As a corollary to the above, during a computation, if q_1 is a member of the initial subset state, then so is q_2.

Example 9. Provide the table of the δ-transition function for the following NFA, and show the computation on i) $w = 010$ and ii) $w = 101$.



2.3 Equivalence between DFA's and NFA's

Since every DFA is also an NFA, it follows that every regular language L is accepted by some NFA. The converse is also true! Every language L that is accepted by an NFA is regular.

This is because an NFA is actually a DFA in disguise. The reason why an NFA is not a DFA is that it does not meet the transition-function definition for DFA's. That definition states

that

$$\delta : Q \times \Sigma \to Q$$

must map a state-input pair to a state. But an NFA N maps a state-input pair to a subset of states. Recalling that N's formal definition is

$$N = (Q, \Sigma, \delta : Q \times \Sigma \to \mathcal{P}(Q), q_0, F),$$

we can make the following changes to this definition to create a DFA N'.

Changing Q to $\mathcal{P}(Q)$: every state is now a subset of the original set of states

Changing the domain of δ: instead of $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ we now have

$$\delta' : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q),$$

which is now a DFA transition function because it maps a subset state-input pair (S, a) to a subset state T.

Changing the initial state: instead of q_0, the initial state is now the singleton set $\{q_0\}$.

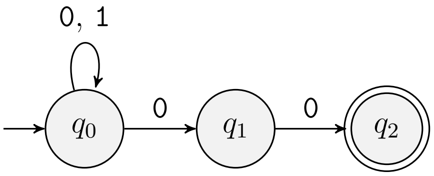
Changing the subset of accepting states: instead of F, we have F' where subset $S \in F'$ iff $S \cap F \neq \emptyset$. In other words, a subset is an accepting state iff it contains some accepting state of N.

The new DFA N' may now be formally expressed as

$$N' = (\mathcal{P}(Q), \Sigma, \delta', \{q_0\}, F').$$

Moreover, the computation of N' on some input word w is defined *exactly* in the same way as the computation of N on w. In other words, N accepts w iff N' accepts w. To see this, as an exercise, write out the definition of what it means to be an accepting computation for a DFA, but within the context of machine N'. Then show that it is identical to the definition of what it means to be an accepting computation for NFA N. Both definitions should appear identical.

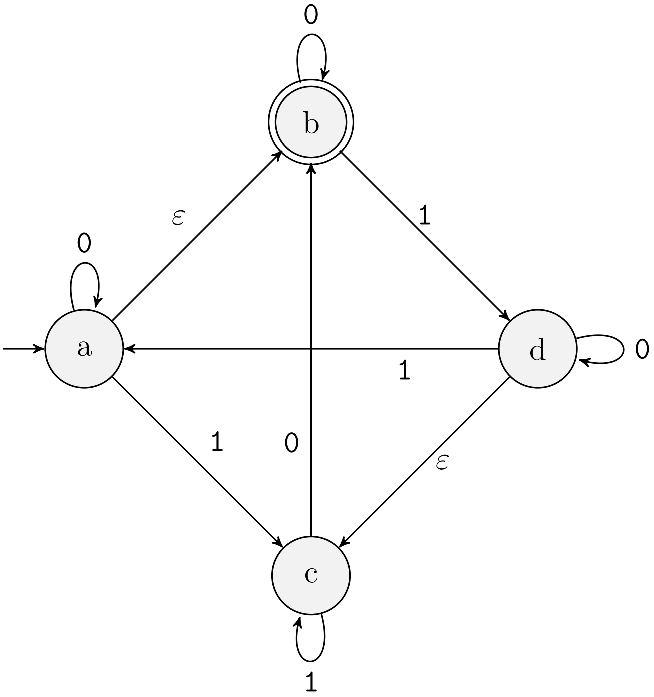
Example 10. Convert the following NFA to a DFA



whose transition function δ is shown below.

$q_i \backslash s$	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\emptyset$
q_2	$\emptyset$	$\emptyset$

Example 11. Convert the following NFA to a DFA by using the method of subset states.



It's transition function is provided in the following table.

$q_i \backslash s$	0	1
a	$\{a, b\}$	$\{c\}$
b	$\{b\}$	$\{c, d\}$
c	$\{b\}$	$\{c\}$
d	$\{c, d\}$	$\{a, b\}$

3 Operations on Languages

Recall that a language is a set of words over some alphabet Σ. For this reason we may apply all the well-known set operations to languages, including union, intersection, complement, difference, and symmetric difference. With the exception of complement which takes only one language, each operation takes two languages, A and B, over some alphabet Σ, and from them produces a third language defined over Σ. In addition the **star** operation takes a language L and produces a new language that is the set of all possible finite concatenations of words from L. Let w be an arbitrary word over Σ. Then,

Complement $w \in \overline{A}$ iff $w \notin A$

Union $w \in A \cup B$ iff either $w \in A$ or $w \in B$

Intersection $w \in A \cap B$ iff $w \in A$ and $w \in B$

Difference $w \in A - B$ iff $w \in A$ and $w \notin B$

Symmetric Difference $w \in A \oplus B$ iff $w \in A$ or $w \in B$, but not both

Concatenation $w \in A \circ B$ iff $w = uv$, where $u \in A$ and $v \in B$

Star $w \in L^*$ iff $w = \varepsilon$, or $w = u_1 u_2 \cdots u_n$, where each $u_i \in L$, $i = 1, 2, \ldots, n$.

Example 12. Let B_0 (respectively, B_1) denote the set of binary strings that begin with a 0 (respectively, 1). Then $B = B_0 \cup B_1$ is the set of binary strings that begin with a 0 or begin with a 1. Compute the words in $\overline{B}$.

Example 13. Let L_2 (respectively, L_3) denote the set of binary strings whose 1-bits sum to a value that is divisible by two (respectively, three). Provide a succinct description of i) $L_2 \cap L_3$, ii) $L_2 \oplus L_3$, iii) $L_3 - L_2$.

Example 14. For L_2 and L_3 in Example 13, provide three words that are in L_2L_3, and three words that are not in L_2L_3. Explain why $L_2L_3 = L_3L_2$.

Example 15. We have the following star languages.

1. $\{0\}^* = \{\varepsilon, 0, 00, 000, \ldots\}$
2. $\{01, 10\}^* = \{\varepsilon, 01, 10, 0101, 0110, 1010, 1001, 010101, \ldots\}$. How many words of length n does this language have if n is even?

Example 16. Consider the language L of all binary strings that begin with two 0's and have an odd number of 1's. Which of the following words are in L^*?

a. 001101001011010010110

b. 0001011001100101010101

c. 00101001101100001100

d. 001100100

4 Closure properties of regular languages

Our goal is now to show that the regular languages are closed under *all* of the language operations defined in this section. For example, if A and B are regular, then are $\overline{A}$, $A \cup B$, $A \cap B$, $A - B$, $A \oplus B$, and A^*. We begin with complement.

Proposition 1. If L is regular, then so is $\overline{L}$.

Proof of Proposition 1. The idea is that, if DFA M accepts L, then a DFA $\overline{M}$ that is equivalent to M, with the exception that a state q is now accepting (respectively, rejecting) for $\overline{M}$ iff it is a rejecting (respectively, accepting) state for M.

More formally, Suppose DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts L. Then DFA $\overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$ accepts $\overline{L}$. This is true since, for any word $w \in \Sigma^*$, $w \in \overline{L}$ iff $\overline{M}$ accepts w, iff the computation of $\overline{M}$ on input w ends in a state belonging to $Q - F$, iff the computation of M on input w ends in a state belonging to $Q - F$, iff $w \notin L$. □

Example 17. Recall L_3 from Example 13. Provide the state diagram of a DFA that accepts $\overline{L_3}$.

4.1 Closure under intersection

$\{a,b\} \times \{1,2,3\} = \{(a,1),(a,2),(a,3),(b,1),(b,2),(b,3)\}$

Proposition 2. If A and B are regular languages over the same alphabet Σ, then so is $A \cap B$.

Proof of Proposition 2. Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts A and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts B. The big-picture idea for why $A \cap B$ is regular is to define a third machine M that makes use of the states and transition function of both M_1 and M_2. Moreover, M performs a computation on some input word w by simultaneously computing *both* the computation M_1 on input w and the computation of M_2 on input w. It accomplishes this by using the Cartesian product $Q_1 \times Q_2$ of the two state sets as its set of states. That way, at any time in the computation, it is simultaneously in both a Q_1-state and a Q_2-state. Furthermore, to make sure that it is accurately following both computations $M_1(w)$ and $M_2(w)$, its transition function δ makes use of both δ_1 and δ_2 as follows: for any $t \in Q_1$, $u \in Q_2$, and $s \in \Sigma$,

$$\delta((t,u),s) = (\delta_1(t,s), \delta_2(u,s)).$$

Finally, M accepts w iff both M_1 and M_2 accept w, which is true iff the final state entered by M belongs to $F_1 \times F_2$, meaning that both $M_1(w)$ and $M_2(w)$ end in accepting states.

More formally, we let

$$M = (\boxed{Q_1 \times Q_2}, \Sigma, \delta, \boxed{(q_1, q_2)}, \boxed{F_1 \times F_2}),$$

where, for any $t \in Q_1$, $u \in Q_2$, and $s \in \Sigma$,

$$\boxed{\delta(\underline{(t,u)},s) = (\underline{\delta_1(t,s)}, \underline{\delta_2(u,s)}).}$$

Then $w \in A \cap B$ iff M accepts $w = w_1 w_2 \cdots w_n$, iff there is a sequence of states

$$(t_0, u_0), (t_1, u_1), \ldots, (t_n, u_n)$$

for which

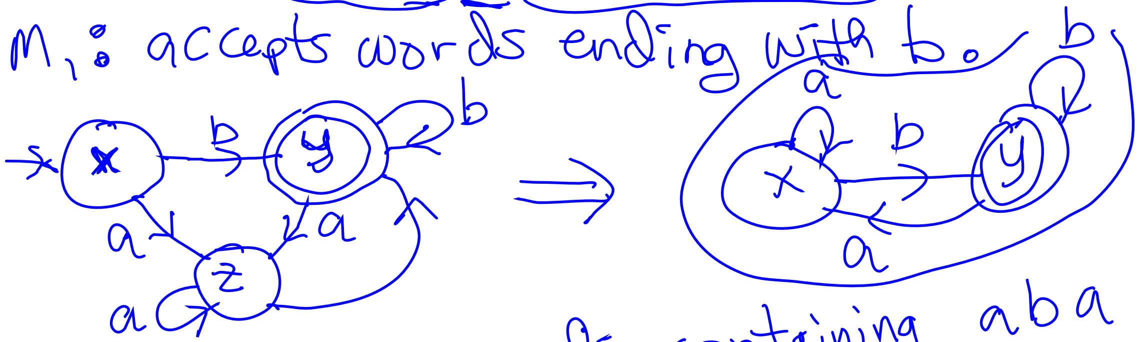
1. $(t_0, u_0) = (q_1, q_2)$ is the initial state,
2. $(t_i, u_i) = \delta((t_{i-1}, u_{i-1}), w_i) = (\delta_1(t_{i-1}, w_i), \delta_2(u_{i-1}, w_i))$, for $i = 1, \ldots, n$, and
3. $(t_n, u_n) \in F_1 \times F_2$

iff $t_0, t_1, \ldots, t_n$ is an accepting computation for M_1 on input w and $u_0, u_1, \ldots, u_n$ is an accepting computation for M_2 on input w, iff $w \in A$ and $w \in B$. □

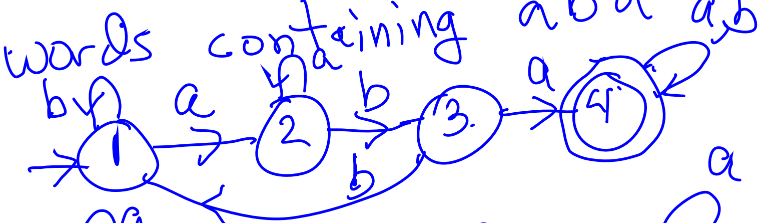
Example 18. Use the construction from Proposition 2 to design a DFA for the language L over $\{a,b\}^*$ consisting of words that end with a `b` and contain the subword `aba`.

DFA M_1: accepts words ending with b.

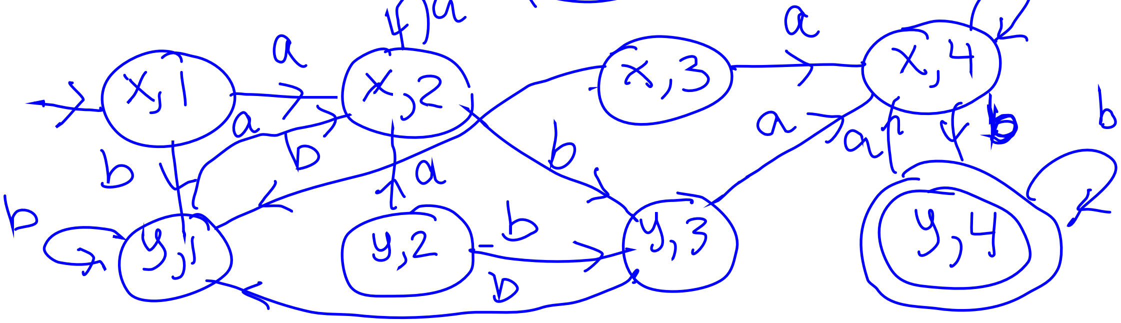
M_1:



DFA M_2: accepts words containing aba



$M_1 \times M_2$:



4.2 Closure under union

If NFA's are equivalent in power to DFA's, then why bother with them? The reason is because many regular languages are more easily defined using an NFA. Especially those languages that are i) a union of two or more regular languages, a concatenation of two regular languages, or ii) the star of some regular language.

Proposition 3. If A and B are regular, then so is $A \cup B$.

Proof of Proposition 3. Suppose NFA M_1 accepts A and NFA M_2 accepts B. Then to construct a state diagram for an NFA N that accepts $A \cup B$, do the following.

1. Construct the state diagram for M_1, where the node labeled with q_1 is its initial state and F_1 is its set of accepting states.
2. Construct the state diagram for M_2, where the node labeled with q_2 is its initial state and F_2 is its set of accepting states.
3. Add a new node labeled with q_0 which serves as N's initial state.
4. Add ε-labeled edges (q_0, q_1) and (q_0, q_2).
5. Designate any state in $F_1 \cup F_2$ as an accepting state for N.

Then, because of the two ε-edges (q_0, q_1) and (q_0, q_2), a computation of N on input word w results in a parallel computation of both M_1 and M_2 on w, where the computation accepts iff either an accepting state of M_1 or an accepting state of M_2 belongs to the final subset state. In other words, $w \in L(N)$ iff $w \in A$ or $w \in B$. □

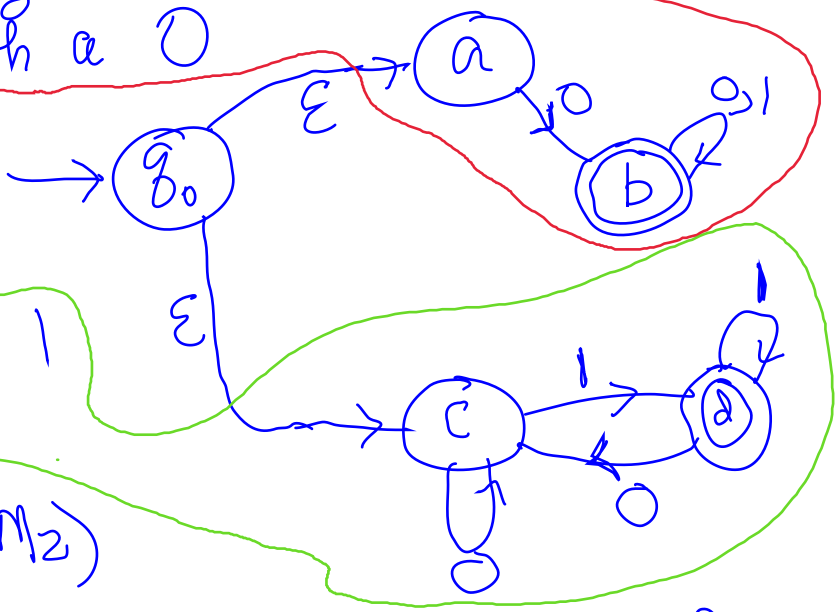
Example 19. Use the procedure described in the Proposition 3 to construct an NFA that accepts all binary words that either being with 0 or end with 1. Show the computation of N on input 101.

begin

M_1: begin with a 0

N: either begin with 0 or end with 1.

M_2: ends with 1



$L(N) = L(M_1) \cup L(M_2)$

$\quad 1 \quad\quad 1 \quad\quad 0 \quad\quad 1$

$\{q_0, a, c\}\ \{d\}\ \{d\}\ \{c\}\ \{d\} \Rightarrow$ accept since d is an accepting state for N.

4.3 Closure under concatenation

Proposition 4. If A and B are regular, then so is $A \circ B$.

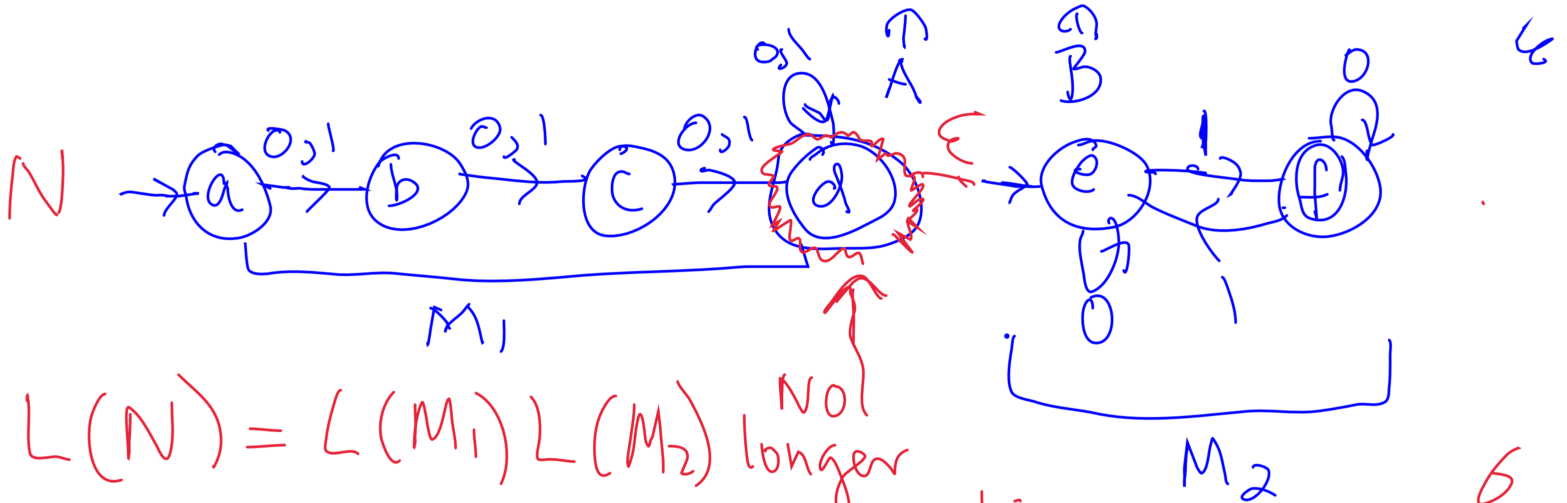
Proof of Proposition 4. Suppose NFA N_1 accepts A, and NFA N_2 accepts B. Then to construct a state diagram for an NFA N that accepts $A \circ B$, do the following.

1. Construct the state diagram for N_1, where the node labeled with q_1 is its initial state and F_1 is its set of accepting states.
2. Construct the state diagram for N_2, where the node labeled with q_2 is its initial state and F_2 is its set of accepting states.
3. For each $q \in F_1$ add the ε-labeled edge (q, q_2).
4. Then the initial state for N is q_1, and F_2 is its set of accepting states.

By adding an ε-edge from an accepting state of N_1 to q_2, it allows for N to accept concatenations of A with B. For example, suppose 1001 is in A, and 0001 is in B. Then the computation of N on 1001 will end with a subset state that not only includes an accepting state of N_1, but also includes q_2, the initial state of N_2. This allows the computation to "start over" and next accept 0001. Thus, the entire word 10010001 will be accepted by N. $\square$

Example 20. Use the procedure described in Proposition 4 to construct an NFA that accepts $A \circ B$, where A is the language of all binary words that have a length of at least three, and B is the language of all binary words that have an odd number of 1's.

$\underbrace{0101}_{\overset{\uparrow}{A}}\ \underbrace{0100}_{\overset{\uparrow}{B}} \in A \circ B$



$L(N) = L(M_1)L(M_2)$

$\overset{0}{a}\quad \overset{1}{b}\quad \overset{0}{c}\quad \overset{1}{\{d,e\}}\quad \overset{0}{\{d,e,f\}}\quad \overset{1}{\{d,e,f\}}\quad \overset{0}{\{d,e,f\}}$

$\overset{0}{\{d,e,f\}}\quad \{d,e,f\} \Longrightarrow$ Accepting since f is accepting for N.

4.4 Closure under star

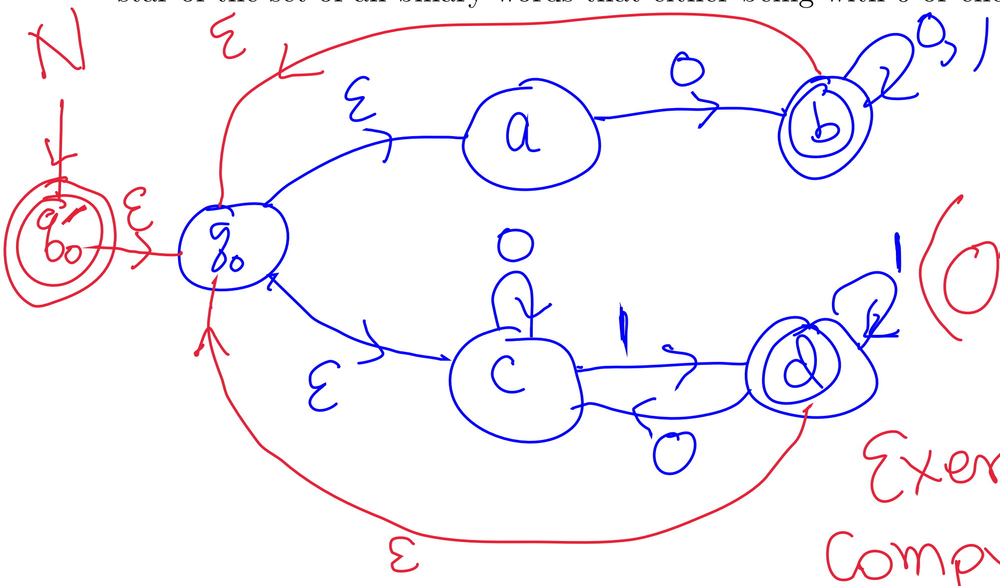
Proposition 5. If A is regular, then so is A^*.

Proof of Proposition 5. Suppose NFA M accepts A. Then to construct a state diagram for an NFA N that accepts A^*, do the following.

1. Construct the state diagram for M, where the node labeled with q_0 is its initial state and F is its set of accepting states.
2. Add a new node labeled with q_0' which serves as N's initial state.
3. Add the ε-labeled edge (q_0', q_0).
4. For each $q \in F$, add the ε-labeled edge (q, q_0).
5. Designate each state in F as an accepting state. Also designate q_0' as an accepting state.

Making q_0' an accepting state takes care of accepting input $\varepsilon \in A^*$. Also, by adding an ε-edge from an accepting state of M to q_0, it allows for N to accept concatenations of words from A. For example, if 1001 and 0001 are both in A. Then the computation of N on 1001 will end with a subset state that not only includes an accepting state of M, but also includes q_0. This allows the computation to "start over" and next accept 0001. Thus, the entire word 10010001 will be accepted by N. More generally, the concatenation of any number of words from A will be accepted by N, and concatenations of words from A are the only words accepted by N (in addition to ε). Therefore, N accepts A^*. $\square$

Example 21. Use the procedure described in Proposition 5 to construct an NFA that accepts the star of the set of all binary words that either being with 0 or end with 1.



$L(N) = \left(0\{0,1\}^* \cup \{0,1\}^*1\right)^*$

Exercise: Show computation on input $\underline{010}\,\underline{101}\,\underline{000}$

5 Regular Expressions

As the name suggests, a regular expression represents a way of providing a function expression (as opposed to a DFA or NFA) in order to represent a regular language. Regular expressions have many applications from desribing tokens in a programming language to providing search criteria for finding data in documents. For example, a line in a text document may be viewed as a single word over some alphabet. A regular expression then provides criteria for the lines that we wish to retrieve from the document.

Let Σ be an alphabet. We provide a structural definition for the set of all legal regular expressions defined over Σ.

5.1 Atomic Regular Expressions

1. For each $\mathtt{a} \in \Sigma$, $\mathtt{a}$ is a regular expression that represents the language $\{\mathtt{a}\}$.
2. ε is a regular expression that represents the language $\{\varepsilon\}$.
3. $\emptyset$ is a regular expression that represents the empty language $\emptyset$.

5.2 Compound Rules for Creating Regular Expressions

1. If R_1 and R_2 are regular expressions and $L(R_i)$ is the language represented by R_i, $i = 1, 2$, then $(R_1 \cup R_2)$ is a regular expression that represents the language $L(R_1) \cup L(R_2)$.
2. If R_1 and R_2 are regular expressions and $L(R_i)$ is the regular language represented by R_i, $i = 1, 2$, then $(R_1 \circ R_2)$ is a regular expression that represents the language $L(R_1) \circ L(R_2)$.
3. If R is a regular expression that represents the language $L(R)$ then (R^*) is also a regular expression that represents the language $L(R)^*$.

5.3 Rules for simplifying regular expression syntax

Dropping Parentheses Parentheses may be omitted when there is no ambiguity. For example, instead of writing (0^*), we may simply write 0^*.

Dropping $\circ$ Concatenation is more commonly expressed by placing side-by-side the two expressions to be concatenated. For example, instead of $0 \circ 1^*$, we may write 01^*. Note: $(01)^*$ would be used to star the expression 01.

Use of { } Given symbols $a_1, a_2 \ldots, a_n \in \Sigma$, instead of writing $a_1 \cup a_2 \cup \cdots \cup a_n$, we may instead write $\{a_1, a_2, \ldots, a_n\}$, or even $(a_1, a_2, \ldots, a_n)$.

+ instead of $*$ L^* allows for the empty string ε. However, sometimes we want the word to have at least one character, in which case we may write L^+, which is defined as $L \circ L^*$. This forces the concatenation of words to have at least one word from L. For example, $\{0,1\}^+$ means all binary words with at least one bit.

$$\{0,1\}\{0,1\}\{0,1\} = \{0,1\}^3$$

Exponentiation For $k \geq 1$ an integer, we may write L^k in place of

$$\underbrace{L \cdots L}_{k \text{ times}} .$$

Example 22. Provide regular expressions for each of the following languages.

a. All binary words that begin with a 0 and end with a 1. $0\{0,1\}^*1$

$$0\,100\,1$$

$$0\,\{0,1\}^*\,1$$

$$100 \in \{0,1\}^*$$

b. All binary words that have an even number of 0's.

$$(1^*01^*01^*)^*$$

$$\{\varepsilon, 00, 0000, \dots\} = L((00)^*)$$

if we do not allow 1s

i.e. $\Sigma = \{0\}$

c. All binary words that have an odd number of 1's and end with a 0.

$$0^*1\underbrace{(0^*10^*10^*)^*}_{\varepsilon}0^+$$

d. All binary words that have a length of at least three and its third bit is 0.

$\{0,1\}^2 0 \{0,1\}^*$

e. The set of words w consisting of a's and b's, and does *not* have the form a^*b^*.

Case 1: a is the first letter

$a^+ b^+ a \{a,b\}^*$

Case 2: b is the first letter

$b^+ a \{a,b\}^*$

Combine the two: $a^* b^+ a \{a,b\}^*$

f. All words in $\{a,b\}^*$ except for aba.

$\{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, abb, baa, bab, bba, bbb\} \cup \{a,b\}^4 \{a,b\}^*$

Example 23. In programming, an identifier is a word that is used to identify some aspect of a program, such as a variable, constant, keyword, function name, the name of a data structure, and the names of its attributes. Consider the following rules for forming an identifier.

Rule 1. must begin with a letter

Rule 2. must end with either a letter or digit

Rule 3. must consist of alphanumeric characters (letters and digits) as well as the dash ("-") character, so long as the dash is immediately preceded (respectively, followed) by an alphanumeric character.

Assuming, an alphabet having letters $\{a, b\}$ and the single numerical digit 0, i) provide a regular expression that describes the set of all legal identifiers and ii) provide a DFA M for which $L(M)$ is the set of all legal identifiers.

Solution. $\mathcal{E} = \{a, b\}\{0, a, b, -0, -a, -b\}^*$

Exercise: Construct a DFA that accepts $L(\mathcal{E})$

Example 24. Syntactically, an integer may be viewed as a sequence of one or more digits that (optionally) follows a negative symbol “-”. We may use this definition to define a number represented in scientific notation. In particular, the number must

Rule 1. (optionally) begin with a negative symbol “-”

Rule 2. begin with (optionally, follwed by) a sequence of zero or more digits

Rule 3. followed by a period “.”

Rule 4. followed by a sequence of zero or more digits

Rule 5. followed by an “e”

Rule 6. followed by an integer.

Rule 7. Rules 2 and 4 cannot both be satisfied by using an empty word.

Assuming, only the digits 0 and 1, i) provide a regular expression that describes the set of all numbers written in scientific notation and ii) provide a DFA M for which $L(M)$ is the set of all numbers written in scientific notation, and that begin with either a digit or a negative symbol.

Case 1: $\{\varepsilon, -\}\{0,1\}^{+} . \{0,1\}^{*} e \{\varepsilon, -\}\{0,1\}^{+}$

Solution.

Case 2: $\{\varepsilon, -\}\{0,1\}^{*} . \{0,1\}^{+} e \{\varepsilon, -\}\{0,1\}^{+}$

Final exp: Case 1 $\cup$ Case 2

Exercise: Write the DFA that accepts this language

$$\{\varepsilon, -\}\left(\{0,1\}^{+} . \{0,1\}^{*} \cup \{0,1\}^{*} . \{0,1\}^{+}\right) e \{\varepsilon, -\}\{0,1\}^{+}$$

6 Regular Expression Languages are Regular

Theorem 1. Let E be a regular expression and $L(E)$ denote the language associated with E. Then $L(E)$ is regular.

Proof of Theorem 1. The proof is by structural induction over the set of all regular expressions. Let Σ denote the alphabet over which E is defined.

Basis step case 1. $E = a$ for some $a \in \Sigma$. Then $L(E) = \{a\}$ is regular since every finite set is regular (prove this!).

Basis step case 2. $E = \varepsilon$. Then $L(E) = \{\varepsilon\}$ is regular (provide the NFA!).

Basis step case 3. $E = \emptyset$. Then $L(E) = \emptyset$ is regular (again, provide the NFA).

Inductive step. Assume E_1 and E_2 are regular expressions for which $L(E_1)$ and $L(E_2)$ are both regular. Then, by Propositions 3,4, and 5, $L(E_1) \cup L(E_2)$, $L(E_1) \circ L(E_2)$, and $L^*(E_1)$, are all regular, where the first is the language associated with $E_1 \cup E_2$, the second the language associated with $E_1 \circ E_2$, and the third the language associated with E_1^*. Therefore, by structural induction, the language associated with an arbitrary regular expression is regular. □

Example 25. Construct an NFA N that accepts $L(E)$ where

$$E = 010\{00, 11\}^*\{10, 01\}.$$

7 Regular Languages are Described by Regular Expressions

The converse to Theorem 1 is also true.

Theorem 2. Every regular language L is associated with some regular expression E, for which $L(E) = L$.

Proof of Theorem 2. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. The goal is to find a regular expression E such that $L(E) = L(N)$. Without loss of generality, we may assume that i) $|F| = 1$ and there is no transitioning from the sole accepting state q_a, and ii) there is no transitioning to initial state q_0 and $q_0 \neq q_a$. Thus N has at least two states.

It turn out that an NFA is a special case of a type of automaton called a **generalized nondeterministic finite automaton (GNFA)**. The state diagram of a GNFA allows for edges to be labeled with regular expressions (either atomic or compound), whereas a transition edge of an NFA is only allowed to be labeled either a finite subset of symbols in Σ, ε, or $\emptyset$ (being labeled with $\emptyset$ is the same as having no transition edge going from some state to another). As a consequence, a GNFA is capable of transitioning from one state q_1 to another q_2 by reading an entire word $w \in L(E)$ where E is the regular expression that labels the edge (q_1, q_2) going from q_1 to q_2. For this reason, a GNFA computation is similar to that of an NFA computation, with the exception that state transitions are triggered by an entire word rather than a single symbol.

We now use induction on the number of states of GNFA N to prove that $L(N)$ is associated with some regular expression.

Basis step. GNFA N has two states. Then $L(N)$ equals $L(E)$ where E is the label of the edge going from q_0 to q_a, and so $L(N)$ is associated with regular expression E.

Inductive step. Assume that every GNFA N having $k \geq 2$ states accepts a language $L(N)$ that is associated with a regular expression E. Now consider a GNFA N having $k+1$ states. The idea is to remove one of N's states q ($q \neq q_0$ and $q \neq q_a$) so that we may use the inductive assumption. We may do this so long as we update the regular expressions on each of the remaining edges of our GNFA's state diagram. For example, consider the edge (q_1, q_2) going from q_1 to q_2 before removing q. Let E denote the regular expression that labels this edge. By removing q, we've lost a way to read a word $w \in \Sigma^*$ that allows us to transition from q_1 to q_2. Namely, any word w that belongs to the language associated with the regular expression

$$E_1 \circ E_2^* \circ E_3,$$

where E_1 is the label for edge (q_1, q), E_2 is the label for the loop (q, q), and E_3 is the label for edge (q, q_2). In other words w is any sequence of symbols that allows one to transition first from q_1 to q, then to loop a finite number of times over q, followed by transitioning from q to q_2. Indeed, if q gets removed, these words could be lost. For this reason we replace the expression E that labels (q_1, q_2) with the new expression

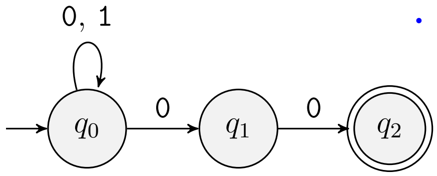
$$E \cup E_1 \circ E_2^* \circ E_3.$$

By doing this, we've preserved all the words that allow us to transition from q_1 to q_2. Moreover, after removing q, the new GNFA N' now has k states, but still accepts the same language as N. Thus, by the inductive assumption, we have

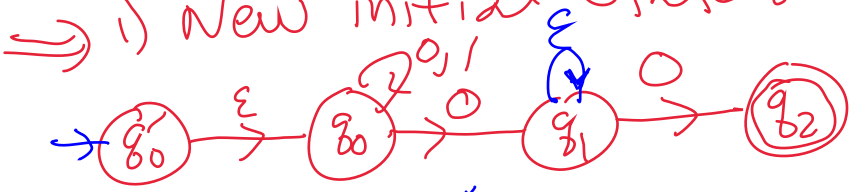
$$L(N) = L(N')$$

is associated with some regular expression. □

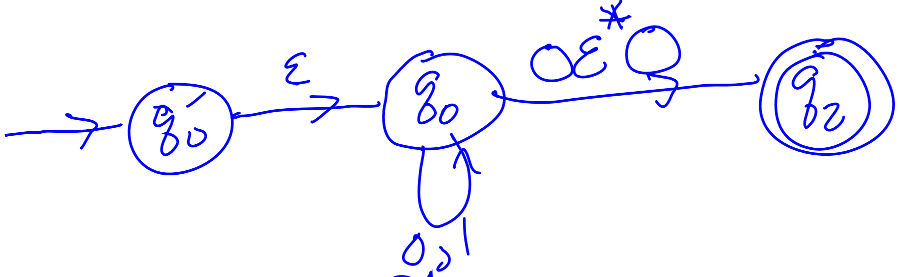
Example 26. Use the proof of Theorem 2 to derive the regular expression associated with the language accepted by the following NFA.



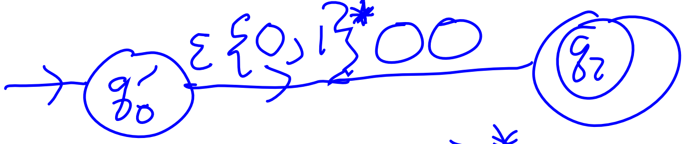
⇒ 1) New initial state :



2) Remove q_1 :



3) Remove q_0 :



4) Final R.E. : $\varepsilon\{0,1\}^*00 \iff \{0,1\}^*00$

Nonregular Languages

Sometimes we may not readily know if a language can be accepted by a DFA. If we suspect that no DFA can accept some language, how can we be sure? The Pumping Lemma can sometimes be used to prove (by way of contradiction) that a language is not regular.

Pumping Lemma If L is a regular language, then there is a positive integer p, called the **pumping length**, such that, if $s \in L$ has length at least p, then s may be divided into three pieces, $s = xyz$ and the following three properties are satisfied.

1. $xy^iz \in L$, for every nonnegative integer i
2. $|y| > 0$
3. $|xy| \leq p$

Proof of the Pumping Lemma. Let M be a DFA that accepts L, and let p be the number of states of M. Note that $p > 0$ since every DFA has at least one state by definition. Let $s \in L$ be such that $|s| \geq p$. Let $q_0, q_1, \ldots, q_p, \ldots, q_{|s|}$ represent the computation of M on input s (note that $q_p = q_{|s|}$ in the event that $|s| = p$, and in any case $q_{|s|}$ is an accepting state). Since $|\{q_0, q_1, \ldots, q_p\}| = p+1$, by the pigeon-hole principle, there exist $0 \leq j < k \leq p$ such that $q_j = q_k$. Thus, when starting at state q_j, the computation of M on subword $s[j+1:k]$ ends back at q_j, since $q_k = q_j$. Let $y = s[j+1:k]$ and define x and z so that $s = xyz$.

The key insight is to notice that, during the computation of M on input s, when the letters of y are read by M, M is in state q_j and M returns to state $q_j = q_k$ after reading the last letter of y. But the same can be said for any positive power y^i: in this case M will start at q_j and return to $q_j = q_k$ upon reading each of the i y's. In other words, M will loop back to q_j i times, once for each y in the string y^i. Hence, starting in state q_j the computation of M on y^i is given by

$$(q_j q_{j+1} \cdots q_k)^i.$$

Similarly, if y were removed from s, then the computation of M on input xz would be

$$(q_0 q_1 \cdots q_j)(q_{k+1} \cdots q_{|s|}).$$

Putting these two observations together yields the fact that the computation of M on input xy^iz is

$$q_0 q_1 \cdots q_{j-1}(q_j q_{j+1} \cdots q_k)^i q_{k+1} \cdots q_{|s|},$$

for each nonnegative integer i. The upshot is that each of these computations end with $q_{|s|}$, an accepting state, and so $xy^iz \in L$ for each nonnegative integer i. For the second property, notice that $|y| = |s[j+1:k]| = k - (j+1) + 1 = k - j > 0$, since $k > j$. Finally, $k \leq p$ which implies $|xy| \leq p$, and the third property is established. QED

Example 27. Use the Pumping Lemma to prove that the language $L = \{0^n 1^n | n \geq 0\}$ is not regular.

Example 28. Let L be the language of binary strings for which there is an equal number of ones and zeros. Show that L is not regular.

Tokenizing a String with DFA's

The use of finite automata and regular expressions is very prominent in the development of language interpreters and compilers. One of the first steps in either task is to tokenize the "source code" that is to be interpreted/compiled. We may think of the source code as a string (or several strings) of characters. By "tokenize", we mean divide the string into words, each of which represents a building block for the program. Furthermore, (for simplicity) we may categorize these token blocks as being one of the following.

Identifier a string of characters that begins with an alphabetical character, followed by alphanumeric characters, including underscore.

Numeric Literal a string of digits that may include a single period character.

String Literal a word over the entire range of ASCII characters delimited by a pair of double-quotation characters.

Operation Literal a word over the alphabet

`{!,@,$,%,^,&,*,-,=,+,|,:,<,>,/,?}`

Delimiter a character from the set

`{[,],(,),{,},,,;}`

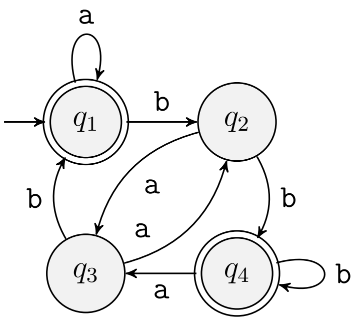
The states of the DFA are then defined as NEUTRAL, READ IDENTIFIER, READ NUMBER, READ STRING, and READ OPERATION. Moreover, when the DFA is in one of the latter four states, it may use a "sub-DFA" to recognize the specific category of token. For example, when the state moves to READ NUMBER, it triggers a DFA that accepts all words that represent a syntactically correct number, and rejects should a character be read that results in a syntactically incorrect number. We leave it as an exercise to provide a DFA for each of the above token categories (excluding Delimiter).

Below we provide a general transition function for the main DFA. This transition table assumes that each character falls into one of a number of non-overlapping sets.

State\Input	ALPHA	DIGIT	OP	SPACE	DELIM	QUOTE
NEUTRAL	IDENTIFIER	NUMBER	OPERATION	NEUTRAL	NEUTRAL	STRING
IDENTIFIER	IDENTIFIER	IDENTIFIER	REJECT	NEUTRAL	NEUTRAL	REJECT
NUMBER	REJECT	NUMBER	OPERATION	NEUTRAL	NEUTRAL	REJECT
STRING	STRING	STRING	STRING	STRING	STRING	NEUTRAL
OPERATION	IDENTIFIER	NUMBER	OPERATION	NEUTRAL	NEUTRAL	STRING

Exercises

1. Consider the following state diagram for a DFA M.



   Provide the following for M.

   (a) start state

   (b) set of accept states

   (c) the sequence of states that M goes through on input `aabb`

   (d) Does M accept `aabb`? ε?

2. Provide a formal definition for the DFA M from the previous exercise.

3. Consider the formal DFA description

$$M = (\{q_1, q_2, q_3, q_4, q_5\}, \{u, d\}, \delta, q_3, \{q_3\}),$$

   where δ is given by the following table.

$q_i \backslash s$	u	d
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_5
q_5	q_4	q_5

   Draw the state diagram for this DFA.

4. For the machine M provided in the previous exercise, provide the computation of M on input ududduddu. Is this input accepted or rejected?

5. Provide the state diagram for a DFA that accepts all words from $\{a, b\}^*$ that have an even number of a's.

6. Use weak induction (on the length n of the input) to prove that the DFA you provided in the previous exercise does in fact accept exactly those words with an even number of `a`'s. Advice: give it your best effort before peaking at the solution.

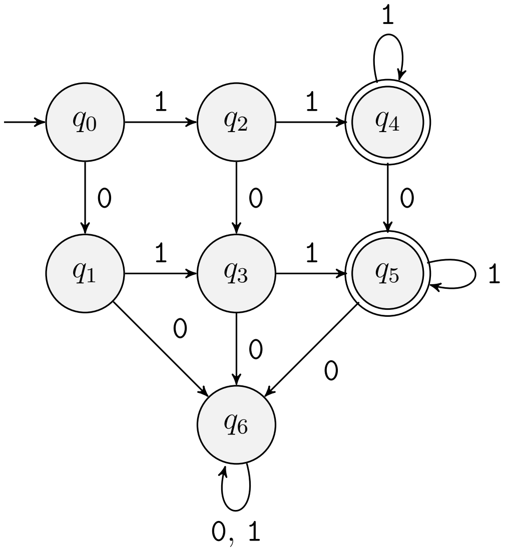
7. Repeat the previous exercise, but now assume the DFA accepts all words from $\{a, b\}^*$ that have an odd number of a's and end with a b.

8. Provide a DFA that accepts all binary strings that contain the substring 0101.

9. Provide a DFA that accepts all binary strings that do *not* contain 0101.

10. Provide a DFA that only accepts ε and 10.

11. Describe the set of binary words that are accepted by the DFA given below. Prove your answer! Hint: consider the structure of the graph, in terms of the paths from initial state to accepting states.



12. Consider the alphabet

$$\Sigma = \left\{ \begin{matrix} 0 \\ 0 \end{matrix}, \begin{matrix} 0 \\ 1 \end{matrix}, \begin{matrix} 1 \\ 0 \end{matrix}, \begin{matrix} 1 \\ 1 \end{matrix} \right\}$$

where each symbol is a column vector of two bits. Provide the state diagram for a DFA that accepts all words w over Σ for which the top layer of w represents a binary number that is greater than the binary number represented by the bottom layer. For example, the DFA should accept

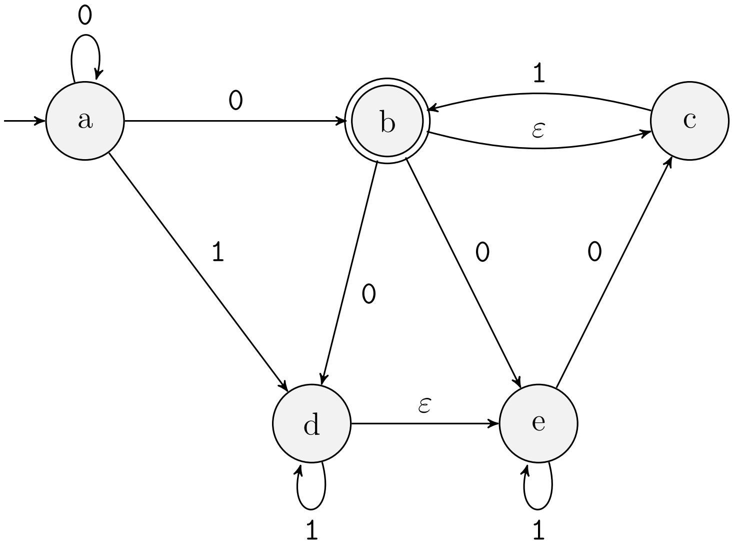
$$w_1 = \begin{matrix} 01011 \\ 00110 \end{matrix}$$

since the top layer represents the number 11, and the bottom layer represents the number 6. However, it should reject

$$w_2 = \begin{matrix} 10001 \\ 10011 \end{matrix}$$

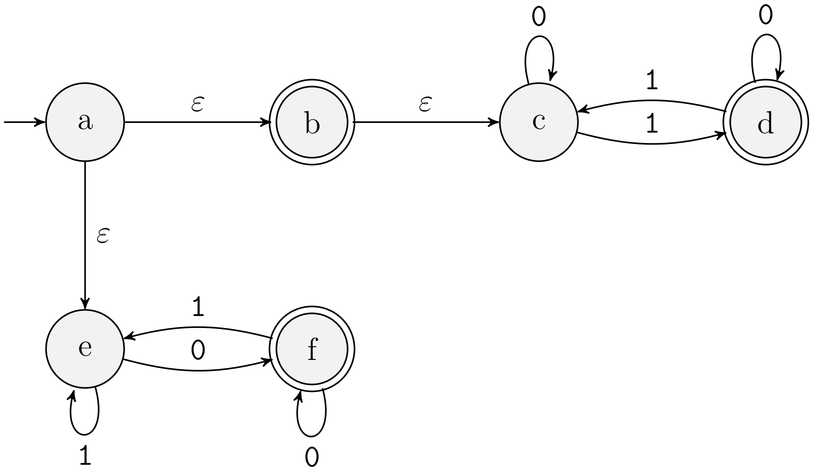
since the top layer represents the number 17, the bottom layer 19, and $17 < 19$.

13. Provide the δ-transition function for the NFA whose state diagram is shown below.



14. For the NFA from the previous exercise, provide the computation on inputs 0010 and 10111. Which of the two are accepted?

15. Provide the δ-transition function for the NFA N whose state diagram is shown below.



16. For the NFA from the previous exercise, provide the computation on inputs ε, 0101 and 000. Verify that all three are accepted by N. Can you find an input that does not get accepted?

17. Provide the state diagram of an NFA that accepts the language

    (a) $\{w|w$ ends with 00$\}$ and uses only three states.

    (b) $\{w|w$ contains the substring 0101$\}$ and uses only five states.

    (c) $\{w|w$ contains an even number of zeros and exactly two ones $\}$ and uses only six states.

    (d) $\{0\}$ and uses only two states.

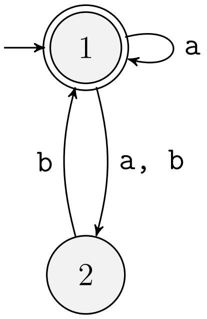
    (e) $0^*1^*0^+$ and uses only three states.

    (f) $1^*(001^+)^*$ and uses only three states.

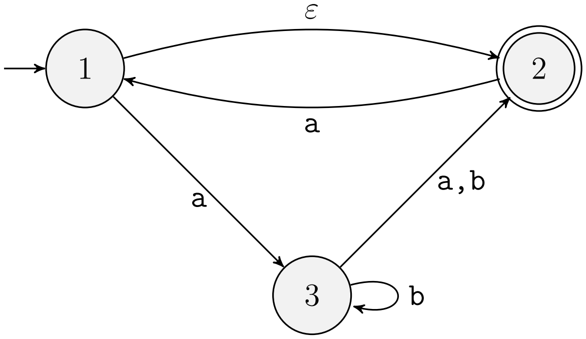
    (g) ε and uses only one state.

    (h) 0^* and uses only one state.

18. Provide a DFA M that accepts the same language as that accepted by the NFA N that is defined in the following state diagram. Do so by defining the states of M to be subsets of the states of N.



19. Provide a DFA M that accepts the same language as that accepted by the NFA N that is defined in the following state diagram. Do so by defining the states of M to be subsets of the states of N.



20. Let L be the regular language of binary strings, where $w \in L$ iff the sum of its 1-bits is divisible by four. Define $\overline{L}$ and provide a DFA that accepts $\overline{L}$.

21. Let L_1 denote the language whose words have exactly one 1 and an odd number of 0's, while L_2 denotes the language whose words have exactly two 1's and an even number of 0's. Provide an example of a word that is i) in both L_1L_2 and L_2L_1, and ii) in L_1L_2 but not in L_2L_1.

22. Which of the following is a member of L^*, where L is the language of binary words that contain at most two 0's and at least three 1's?

    (a) 10100011011101

    (b) ε

    (c) 010111110111000111

    (d) 110010101011100

23. Let L be the regular language $\{\texttt{abaa}, \texttt{aa}\}$ defined over $\{\texttt{a}, \texttt{b}\}^*$. Define $\overline{L}$ and provide a DFA that accepts it.

24. Which of the following is a member of L^*, where L is the language $\{\texttt{aba}, \texttt{bbaa}\}$?

    (a) `abaababbbaa`

    (b) `ababbaa`

(c) `bbaababbaba`

(d) `bbaabbaaabaababbaa`

25. Let L denote the set of binary strings w, such that $|w| \geq 3$ and the last bit of w equals 1. Use the construction from the proof of Proposition 2 to design a DFA for L, where we assume $L = A \cap B$, where A is the set of binary strings having length at least three, and B is the set of binary strings that end with a 1. First draw the DFA's that decide A and B, and then use them to design the DFA that decides L.

26. Repeat the previous exercise, but now assume $w \in L \subseteq \{a, b\}^*$ iff w has an odd length and an even number of `a`'s.

27. Use the fact that regular languages are closed under complement and intersection to prove that they are closed under i) union, ii) difference, and iii) symmetric difference. Hint for i): use De Morgan's rule.

28. Provide a regular expression that describes each of the following binary-word languages.

    (a) all words that begin with a 1 and end with a 0

    (b) all words that contain at least 3 1's

    (c) all words that contain 0101

    (d) all words having length at least 3 and whose third bit is a 0

    (e) all words that either start with a 1 and have odd length, or start with a 0 and have even length

    (f) all words that do *not* contain substring 110

    (g) all words having length at most 5

    (h) all words except for 11 and 111

    (i) all words for which every odd bit is a 1

    (j) all words that have at least two 0's and at most one 1

    (k) the language consisting of ε and 0

    (l) words that contain an even number of 0's or (inclusive) contain exactly two ones

    (m) the language that has no words

    (n) all words except the empty string

29. For each of the following regular expressions, provide an NFA that accepts the language described by the regular expression.

    (a) $a(abb)^* \cup b$

    (b) $a^* \cup (ab)^*$

    (c) $(a \cup b^*)a^*b+$

30. Exercise 1.9 from ITC

31. Exercise 1.10 from ITC

32. Exercise 1.11 from ITC

33. Exercise 1.12 from ITC (Just give the DFA)

34. Exercise 1.13 from ITC

35. Exercise 1.14.b from ITC

36. Exercise 1.15 from ITC

37. Exercise 1.16 from ITC

38. Recall that a palindrome is a word that reads the same forwards as backwards. Provide a structural definition for the set of all binary palindromes.

39. Use the previous exercise and structural induction to prove that any binary palindrome having even length must have an even number of ones.

40. Provide a structural definition for the language associated with the regular expression a*b*.

41. Recall that a full binary tree is a tree for which each internal node (i.e. a node with at least one child) has exactly two children. Consider the following structural definition for the set of binary encodings of full binary trees. The word 0 is a binary encoding of the full binary tree consisting of a single node. Now suppose T_1 and T_2 are full binary trees, where w_i is the binary encoding of tree T_i, $i = 1, 2$. Then $0w_11w_2$ is the binary encoding of the tree that consists of a root whose left subtree is T_1 and whose right subtree is T_2. Draw the binary tree associated with each of the following binary encodings.

    (a) 0010010010

    (b) 0001010010

    (c) 0001001010

42. Recall the set of binary encodings of full binary trees that was defined in the previous exercise. Prove or disprove: for each binary encoding there is a unique binary tree (up to isomorphism) associated with that encoding. By "unique up to isomorphism", we mean that two trees having the same binary encoding would have to look exactly the same. Thus, to disprove the statement, you must find two full binary trees that look different, but have the same encoding.

43. Use structural induction to prove that all well-formed sequences of parentheses (as defined in lecture) must begin with a left parenthesis and end with a right parenthesis.

44. Exercise 1.17 from ITC

45. Provide a regular expression whose associated language is the set of all binary words for which each odd-numbered bit equals 1.

46. Exercise 1.19 from ITC

47. Exercise 1.20 from ITC

48. Exercise 1.21 from ITC

49. Exercise 1.22 from ITC

50. Exercise 1.23 from ITC

51. Exercise 1.28 from ITC

52. Exercise 1.29 from ITC

53. Exercise 1.30 from ITC

54. An identifier for some programming language is a word that begins with a letter and consists of both underscore and alphanumeric characters, where two consecutive underscore characters are not allowed, and the last character should not be an underscore. Provide a regular expression whose associated language is the set of all possible identifiers. For simplicity, assume $\Sigma = \{a, b, 0, 1, \_\}$.

55. Scientific notation format has the form

$$ab.c_1 \cdots c_m S d e_1 \cdots e_n,$$

where $a, d \in \{-, \varepsilon\}$, b is a positive digit, $c_1, \ldots, c_m, e_1, \ldots, e_n$ are digits, and $S \in \{\text{e,E}\}$. For example, the number -0.001394 can be written as

$$-1.394\text{E} - 3.$$

Provide a regular expression whose associated language is the set of all possible scientific-notation words. For simplicity, assume $\Sigma = \{0, 1, 2, -, e, E, .\}$.

Exercise Solutions

1. We have the following.

   (a) q_1

   (b) $F = \{q_1, q_4\}$

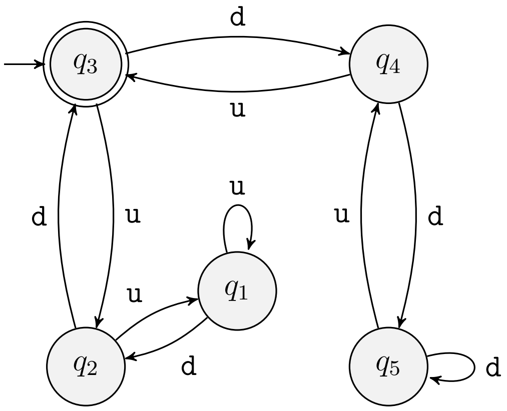
   (c) q_1, q_1, q_1, q_2, q_4

   (d) M accepts aabb since the final state of the computation $M(\texttt{aabb})$ is $q_4 \in F$. M accepts ε since the final state of the computation $M(\varepsilon)$ is $q_1 \in F$.

2. $M = (\{q_1, q_2, q_3, q_4\}, \{a, b\}, \delta, q_1, \{q_1, q_4\})$ where δ is defined by the following table.

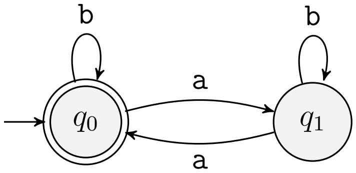
$q_i \backslash s$	a	b
q_1	q_1	q_2
q_2	q_3	q_4
q_3	q_2	q_1
q_4	q_3	q_4

3. We have the following state diagram.



4. $q_3, q_2, q_3, q_2, q_3, q_4, q_3, q_4, q_5, q_4$ is a rejecting computation.

5. We have the following state diagram.

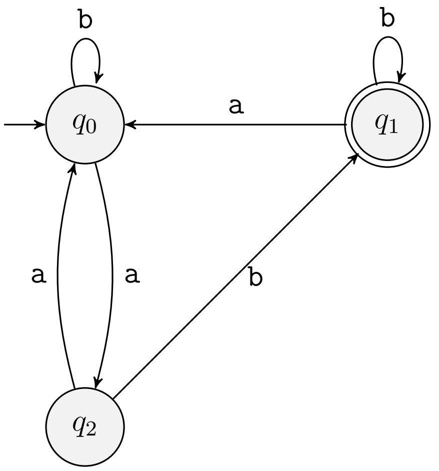


6. **Basis Step:** Let w be an input of length 0. Then $w = \varepsilon$ and the computation ends in accepting state q_0 which is desired, since ε has 0 a's, and 0 is an even number. **Inductive Step:** Assume that, any word w, where $|w| = n$, M accepts w iff w has an even number of a's. Show that for any w with of length $n + 1$ gets accepted iff w has an even number of a's. Since $|w| \geq 1$, we may write $w = w'x$, where w' is a word of length n and $x \in \{\texttt{a,b}\}$.

   Case 1. w' has an even number of a's and $x = \texttt{a}$. Then by the inductive assumption, the computation on input w' ends in q_0, and then reading x moves it to q_1 which is rejecting. This is corrrect since $w = w'x$ has an odd number of a's

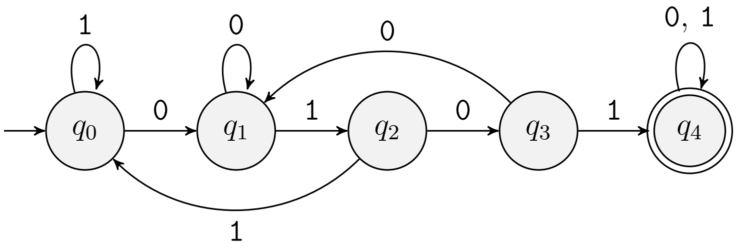
Cases 2, 3, and 4 are similar. Write them!

7. We have the following state diagram.

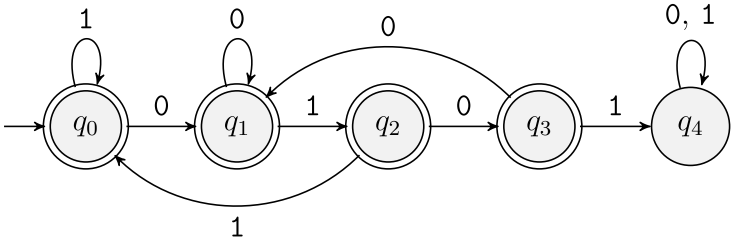


q_0: even number of a's; q_1: odd number of a's and ending with b; q_2: odd number of a's and ending with a;

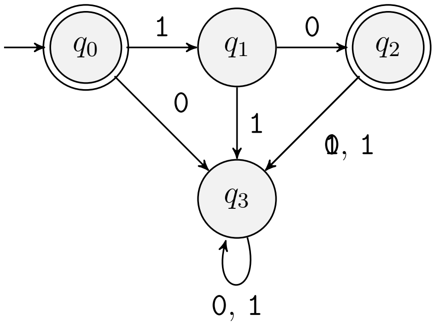
8. We have the following state diagram.



9. We have the following state diagram.



10. We have the following state diagram.



11. The DFA accepts all words that have at least two 1's and at most one 0. **Proof.** Notice that accepting-state q_4 can only first be reached via the input prefix 11, while accepting-state q_5 can only first be reached either by input prefex 011 or 101. Thus, upon first reaching either accepting state with an input word, that word must have a prefix with at least two 1's and at most one 0. Moreover, any additional number of 1's keeps the computation in either q_4 or q_5. The only way to (permanently) escape both of these states is when a second 0 gets read.

12. Let $M = (\{\text{a,b,c}\}, \Sigma, \delta, \text{a}, \{\text{b}\})$, where Σ is defined by the problem statement. State a: the numbers read so far are equal, b: the top number is greater than the bottom, c: the bottom number is greater than the top. The following table provides δ.

$Q\backslash\Sigma$	0	0	1	1
	0	1	0	1
a	a	c	b	a
b	b	b	b	b
c	c	c	c	c

Once the top number has been determined to be greater (respectively, smaller) than the bottom number, it enters the permanent state b (respectively, c).

13. The following table provides the $\delta(Q, \Sigma)$ transition function.

$Q\backslash\Sigma$	0	1
a	{a,b,c}	{d,e}
b	{d,e}	$\emptyset$
c	$\emptyset$	{b,c}
d	$\emptyset$	{d,e}
e	$\{c\}$	$\{e\}$

14. We have the following computations.

Input Symbol Read	**Current Subset State**
0	$\{a\}$
0	{a,b,c}
1	{a,b,c,d,e}
0	{b,c,d,e}
Rejecting State:	{c,d,e}

Input Symbol Read	**Current Subset State**
1	$\{a\}$
0	{d,e}
1	{c}
1	{b,c}
1	{b,c}
Accepting State:	{b,c}

15. The following table provides the $\delta(Q, \Sigma)$ transition function.

$Q\backslash\Sigma$	0	1
a	$\emptyset$	$\emptyset$
b	$\emptyset$	$\emptyset$
c	{c}	{d}
d	{d}	{c}
e	{f}	{e}
f	{f}	{e}

16. We have the following computations. For ε, the final state equals {a,b,c,e} which is accepting since b is an accepting state. Think about it in the following way. When a processor begins its computation by placing a pebble at state a, the ε-transitions allow to also place pebbles at states b, c, and e before reading any input. Hence, for all practical purposes, the initial state is actually {a,b,c,e}.

Input Symbol Read	**Current Subset State**
0	$\{a, b, c, e\}$
1	{c,f}
0	{d,e}
1	{d,f}
Rejecting State:	{c,e}

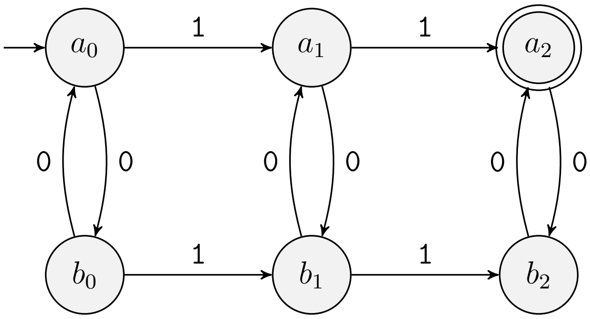
Input Symbol Read	**Current Subset State**
0	$\{a, b, c, e\}$
0	{c,f}
0	{c,f}
Accepting State:	{c,f}

17. We have the following.

   (a)

   (b)

   (c) We have the following NFA.



   (d)

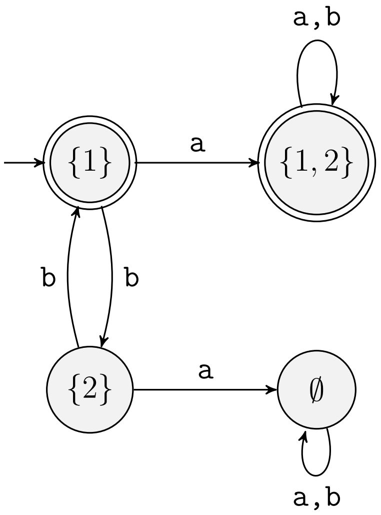
   (e)

   (f)

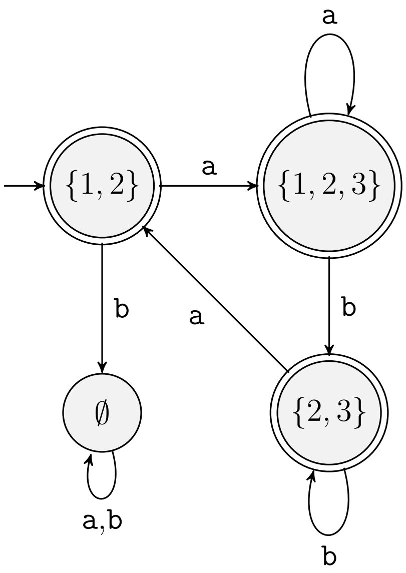
   (g)

   (h)

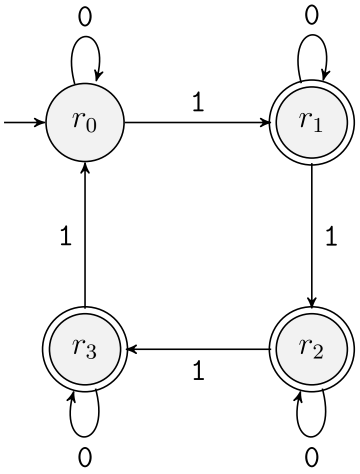
18. We have the following DFA.



19. We have the following DFA. Notice that the initial state is $\{1, 2\}$, since, becuase of the ε edge, (initially) entering state 1 triggers the entering of state 2.



20. $\overline{L}$ is the set of all binary words w, the sum of whose 1-bits is *not* divisible by 4.



21. We have $01001001 \in L_1L_2 \cap L_2L_1$ since we may write it both as (01)(001001) and as (010010)(01). However, $0111 \in L_1L_2 - L_2L_1$ since (01)(11) is the only valid parsing.

22. We have the following results.

    (a) 10100011011101: no. There must be some prefix of this word that is a member of L. But 10100 cannot be a prefix of *any* word in L since it has three zeros and only two ones.

    (b) ε: yes. $\varepsilon \in L^*$ for any language L.

    (c) 01011110111000111: yes.

    $$01011110111000111 = 0101111 \cdot 01110 \cdot 00111,$$

    all three of which are words in L.

    (d) 110010101011100: no. If this word were in L^*, then it would have to be a concatenation of words from L, where the first word would have to be 11001, since the next bit is a zero, which would mean too many zeros. But this is followed by 01010 and no prefix of this word can be in L. Nor can this word be a prefix of some word in L.

23. Similar to Exercise 20: construct the DFA for L and turn accepting (respectively, rejecting) states into rejecting (respectively, accepting) states.

24. We have the following.

    (a) `abaababbaa`: yes.

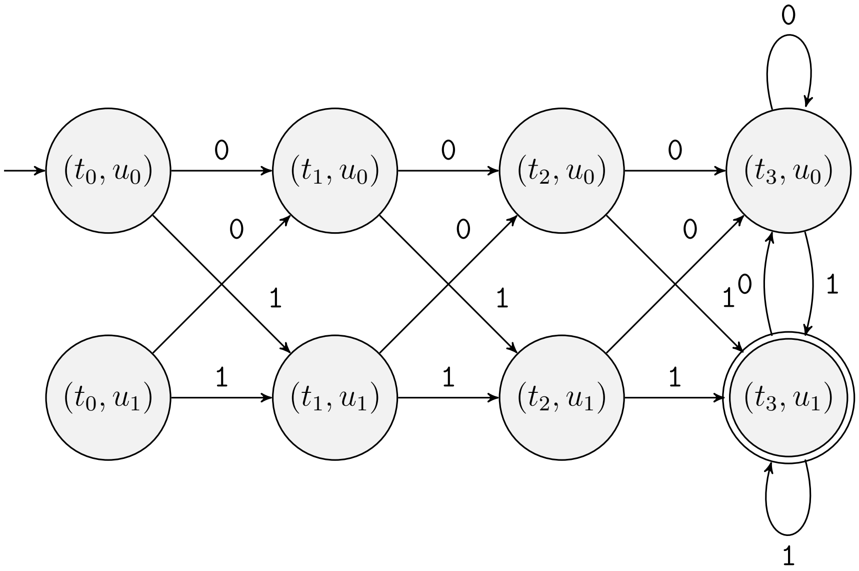
    (b) `ababbaa`: yes.

    (c) `bbaababbaba`: no. The first word in the concatenation would have to be `bbaa`. But then there is no word in L that begins with `ba`, which is what comes next.

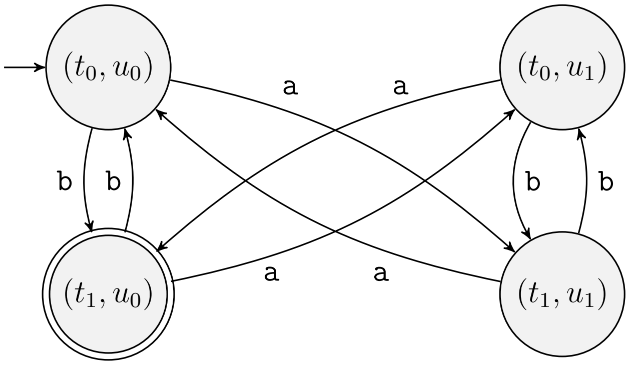
    (d) `bbaabbaaabaababbaa`: yes.

    $$\texttt{bbaabbaaabaababbaa} = \texttt{bbaa} \cdot \texttt{bbaa} \cdot \texttt{aba} \cdot \texttt{aba} \cdot \texttt{bbaa}.$$

25. We have the following. Notice that state (t_0, u_1) is unreachable, and so can be omitted. t_i: a length-i word has been read, $i = 0, 1, 2, 3$. u_i: a word ending with i has has been read, $i = 0, 1$.



26. We have the following. t_0: a word of even length has been read. t_1: a word of odd length has been read. u_0: an even number of a's have been read. u_1: an odd number of a's have been read.



27. $A - B = A \cap \overline{B}$ is regular since A and B are both regular, and regular languages are closed under complement and intersection.

28. We have the following regular expressions.

   (a) $1\{0,1\}^*0$

   (b) $\{0,1\}^*1\{0,1\}^*1\{0,1\}^*1\{0,1\}^*$

   (c) $\{0,1\}^*0101\{0,1\}^*$

   (d) $\{0,1\}\{0,1\}0\{0,1\}^*$

   (e) $1\{00,01,10,11\}^* \cup 0\{00,01,10,11\}^*\{0,1\}$

   (f) Notice that once a word has two consecutive 1's, then all subsequent bits must equal 1. Thus, there are two cases: w contains only 0's or is empty, or w contains 1 or more isolated 1's, and ends with zero or more 1's. Therefore, we have

$$0^* \cup \{\varepsilon, 1\}(0^+1)^*0^*1^*$$

   (g) $\bigcup_{i=0}^{5}\{0,1\}^i$, where, e.g., $\{0,1\}^3$ is shorthand for $\{0,1\}\{0,1\}\{0,1\}$ and $\{0,1\}^0 = \{\varepsilon\}$.

(h) We have

$$\{\varepsilon, 0, 1, 00, 01, 10, 000, 001, 010, 011, 100, 101, 110\} \cup \{0,1\}\{0,1\}\{0,1\}\{0,1\}\{0,1\}^*,$$

where the first expression (to the left of the union) includes all words, except 11 and 111, having length less than 4, while the second expression represents all binary words of length at least 4.

(i) There are three cases depending on whether w has even or odd length:

$$\{10, 11\}^* \cup 1\{01, 11\}^*.$$

(j) $000^* \cup 000^*10^* \cup 0^*1000^* \cup 00^*100^*$

(k) $\{\varepsilon, 0\}$ assuming the convention that the former is the same as $\varepsilon \cup 0$

(l) $(1^*01^*0)^*1^* \cup 0^*10^*10^*$

(m) $\emptyset$

(n) $\{0,1\}^+$

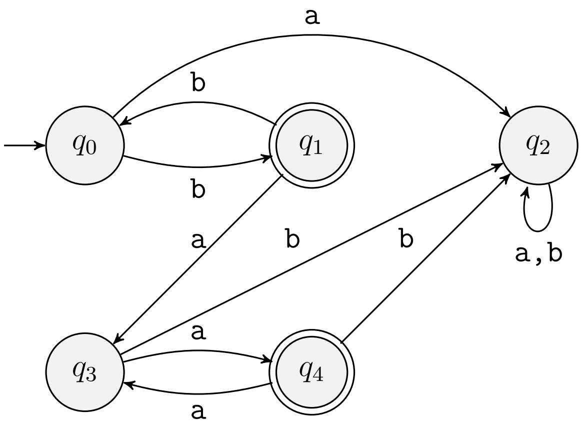
29.

30. Exercise 1.9 from ITC

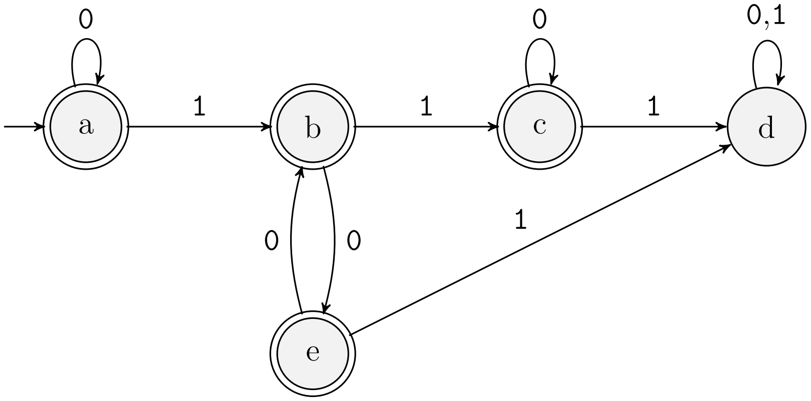
31. Exercise 1.10 from ITC

32. Exercise 1.11 from ITC

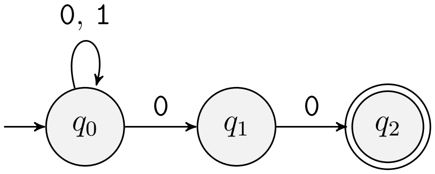
33. Notice that, once the first `a` is read, reading a subsequent `b` leads to rejection. Thus, if an even number of `b`'s had previously been read (state q_0) then reading the first `a` results in rejection. Otherwise (current state is q_1) begin counting the `a`'s and accept only if there are an even number of them, and no `b`'s have subsequently been read.



34. The key insight is to notice that at most two 1's can be read during an accepting computation. For suppose three 1's have been read. Then input word w has the subword 10^m10^n1, where m and n are even. But this means that the first and last 1 are separated by $m + n + 1$ symbols, which is odd. Thus, we have the following DFA.



35. Consider the NFA N below, and let L be the language accepted by N.

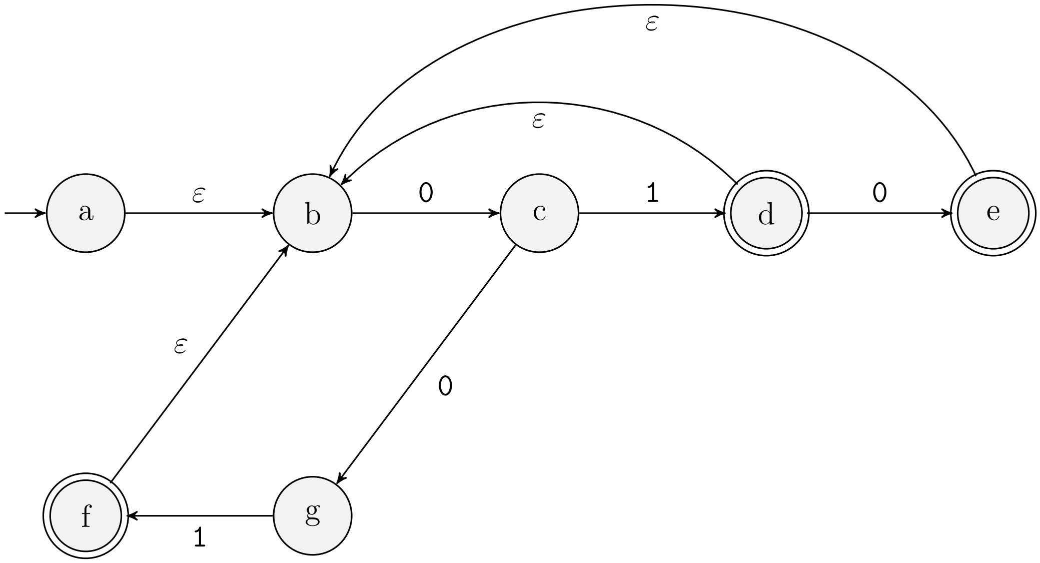


This NFA accepts 00 and the accepting computation has the accepting subset state of $\{q_0, q_1, q_2\}$ (verify this!). Thus, $00 \notin \overline{L}$. Now, if we design a new NFA N' by simply making q_0 and q_1 accepting states, while making q_2 a rejecting state, then N' will *still* accept 00, since the final subset state will again equal $\{q_0, q_1, q_2\}$ which has two accepting states. Therefore, simply swapping states is insufficient for creating an NFA that accepts the complement of the language accepted by another NFA.

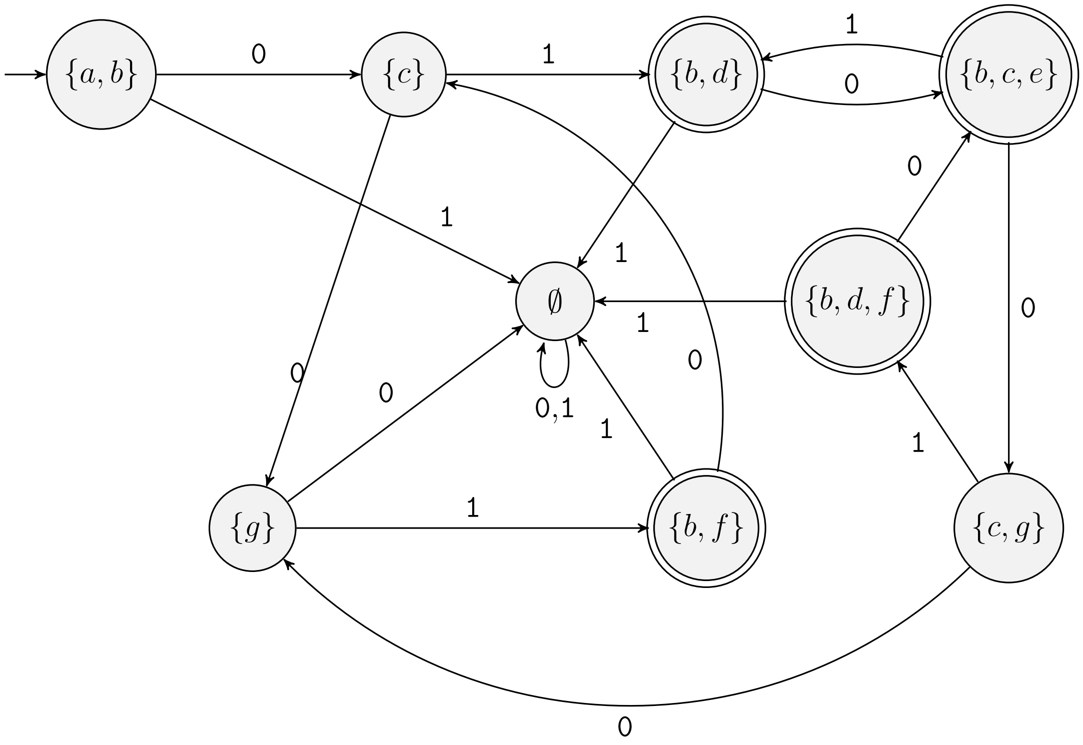
36. Exercise 1.15 from ITC

37.

38. (a) We have the following NFA state diagram.



(b) We have the following DFA state diagram.



39.

40.

41.

42.

43.

44.

45. $1\{01, 11\}^*$

46. Exercise 1.19 from ITC

47. Exercise 1.20 from ITC

48. Exercise 1.21 from ITC

49. Exercise 1.22 from ITC

50. First Assume that $B = B^*$. Then we have

$$BB \subseteq B^* \subseteq B,$$

where the first inclusion is from the definition of B^*, while the second one is by our assumption.

Now assume, $BB \subseteq B$. We already know that $B \subseteq B^*$. Now we must show that $B^* \subseteq B$. Recall that B^k is the set of all words w for which $w = u_1 u_2 \cdots u_k$, where each $u_i \in B$. In other words, it is the language of words that consist of k concatenations of words from B (note:

$B^0 = \{\varepsilon\}$). We use induction to prove that $B^k \subseteq B$, for all $k \geq 0$. This will prove the statement since $B^* = B^0 \cup B^1 \cup \cdots B^k \cup \cdots$.

Basis Step. $B^1 = B \subseteq B$, and $B^2 = BB \subseteq B$, by assumption. What about $B^0 = \{\varepsilon\}$? Show that $B^2 \subseteq B$ implies $\varepsilon \in B$.

Inductive Step.